系統晶片設計
SOC Design

LAB4-1

組員:郭銘宸、郭紘碩

學號:311511082、311512065

系所:電機所碩二、電控所碩二

一、 Explanation of your firmware code

(1) How does it execute a multiplication in assembly code?

在 fir 中所使用的乘法器轉換到組合語言裡首先會先宣告一個 global function，並且編譯系統將它命名為__mulsi3，當組合語言的進度到了要使用乘法器時，便會以呼叫函式的方式將乘法器叫出來使用，而呼叫函數這樣的動作可能表示在組合語言中乘法器不是一個單純的乘法電路硬體，而是可能牽涉到多個有號數運算等指令來完成，因此編譯系統會根據需求產生適當的乘法函式提供呼叫。

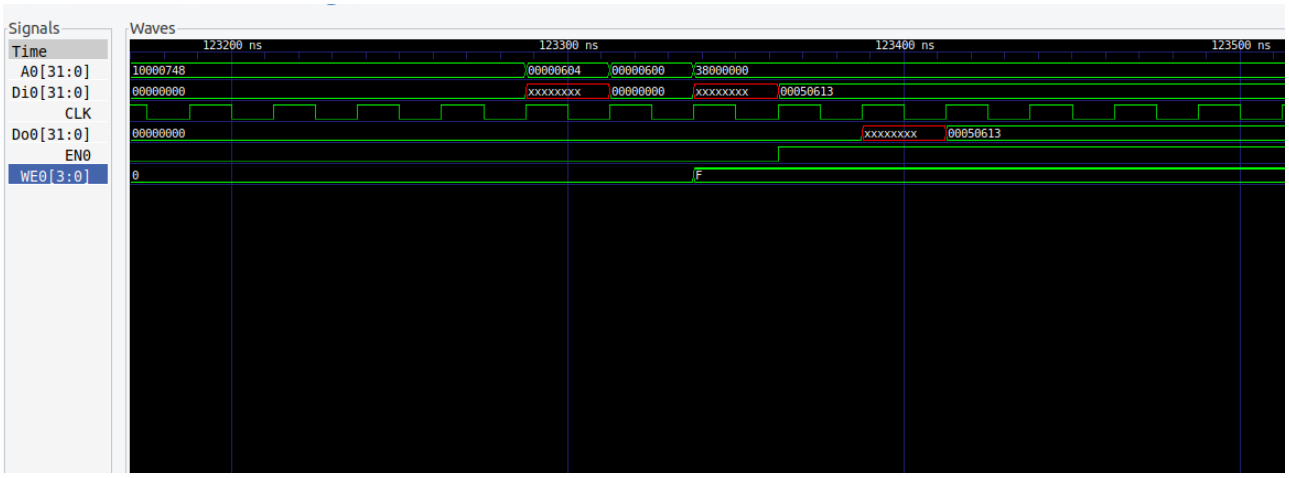
```
102   .LBE2:
103       .loc 1 8 1
104       nop
105       nop
106       lw  s0,28(sp)
107       .cfi_restore 8
108       .cfi_def_cfa 2, 32
109       addi    sp,sp,32
110       .cfi_def_cfa_offset 0
111       jr  ra
112       .cfi_endproc
113   .LFE0:
114       .size   initfir, .-initfir
115       .globl  __mulsi3
116       .align  2
117       .globl  fir
118       .type   fir, @function
```

```
191   .L8:
192       .loc 1 33 29 discriminator 2
193       lui a5,%hi(taps)
194       addi    a4,a5,%lo(taps)
195       lw  a5,-28(s0)
196       slli    a5,a5,2
197       add a5,a4,a5
198       lw  a5,0(a5)
199       .loc 1 33 23 discriminator 2
200       lw  a1,-24(s0)
201       mv  a0,a5
202       call    __mulsi3
203       mv  a5,a0
204       mv  a4,a5
205       .loc 1 33 12 discriminator 2
206       lw  a5,-20(s0)
207       add a5,a5,a4
208       sw  a5,-20(s0)
209       .loc 1 25 47 discriminator 2
210       lw  a5,-28(s0)
211       addi    a5,a5,-1
212       sw  a5,-28(s0)
```

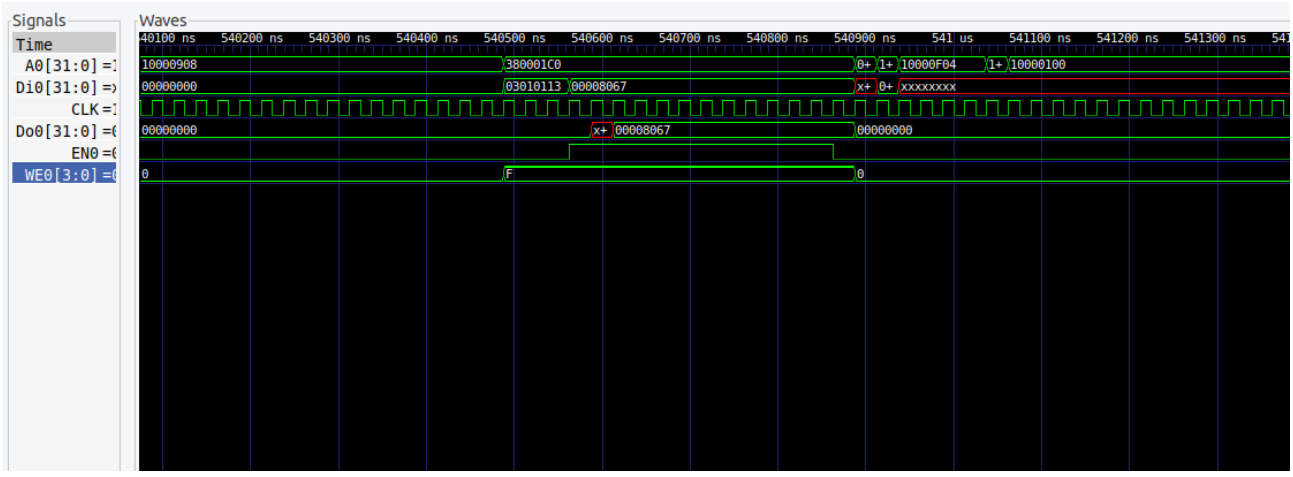
(2) What address allocate for user project and how many space is required to allocate to firmware code?

在下圖(一)中能看到 38000000 為 firmware code 的起始放置位置，而地址的出現每次會帶來 WE，並且當 EN 和 WE 都為 1 時會在下個 clk 寫入資料 Di，隨著 clk 的增加直到 firmware 被完全放入 user bram，能夠從圖(二)中看到最後的放置位置為 380001C0，這樣的起始到結束使用了 4'h01C0 的空間，相當於十進位的 0 ~ 448 的空間。

考慮到 bram 每次儲存位置都是以 word (4 bit)為單位位移，因此總共有 112 個 bram 的 32bit 資料被使用來儲存，在這樣的情況下總共使用 3584bit 來存放 firmware code，相當於 3.58KB，而當我在 windows 中察看.hex 檔案大小時，其顯示的大小為 7KB，所以對於檔案大小的不能純粹靠系統中顯示內容來看，應該看他在 ram 中實際存放的行為來進行計算。



圖(一)

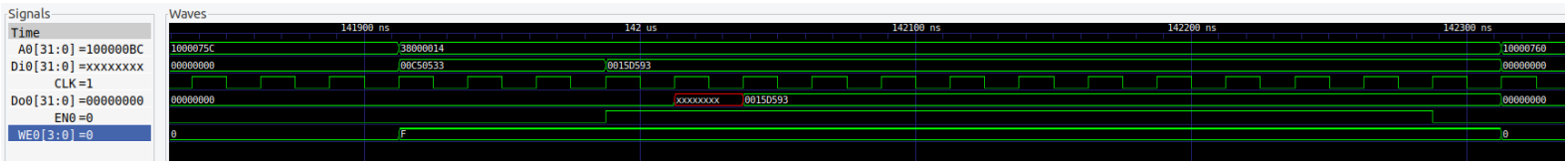


圖(二)

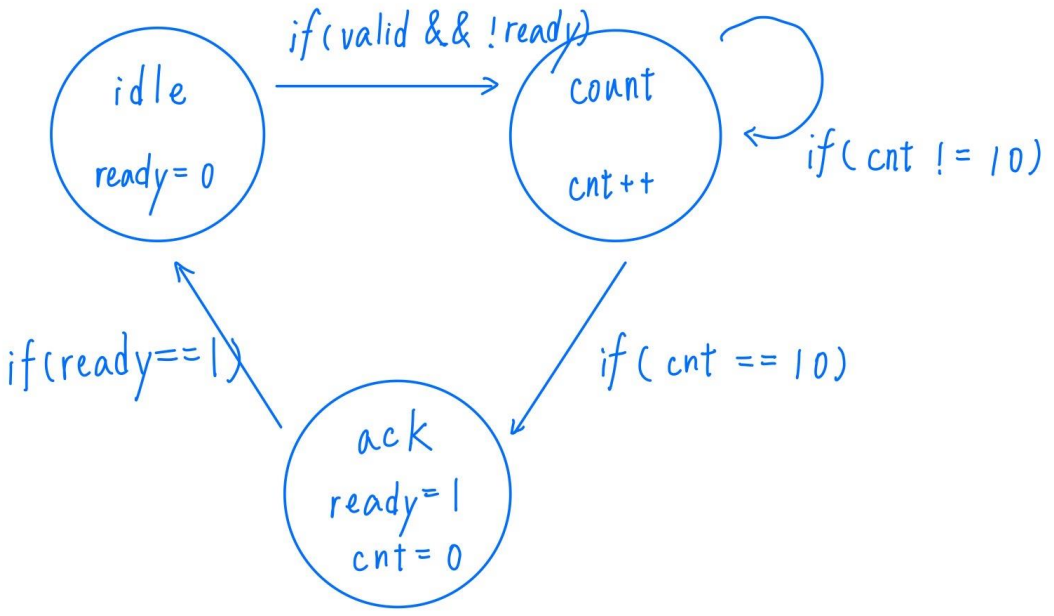
二、 Interface between BRAM and wishbone

(1) Waveform

從下圖 BRAM 波形圖中能看到他與 wishbone 之間式是如何傳遞資料，藉由觀察這個 Interface 的工作方式，能發現當到波形中的 EN 訊號為 1 時，在下個 clk 所讀到的 Do 就是 RAM 地址中的資料，再下一個 clk 之後就會將輸入 Di 寫入 RAM[addr]中，為了使 wishbone 所傳來的資料符合 BRAM 的運作，其中的 EN 訊號是由 wbs_cyc_i && wbs_stb_i 所組合產生出來的，而 WE 訊號則是藉由 wbs_sel_i & {4{wbs_we_i}} 產生的，輸入、地址則是分別把 wbs_dat_i 和 wbs_adr_i 直接接在 BRAM 的 port 上即可，其中比較值得注意的是 wbs_cyc_i 是計算了資料傳過來後藉由 counter 數了 10 個 cycle 後才將 ready 設為 1，也就是 wbs_ack_o 會延遲 10 個 cycle 才升起表示完成一次傳遞，透過將 wishbone 中的這些訊號結合便能夠正確讓 BRAM 運作。



(2) FSM



上圖為 wishbone 和 bram 間介面的狀態圖，當處於 idle 狀態時 ready 訊號總是設為 0，而 valid 升起時會進到 counter 狀態，在這個狀態中會數 10 個 cycle，直到 counter 數滿足 delay 時便會進到下一個狀態，在下一個 ack 狀態中，會將 ready 設為 1 並且歸零 counter 值，隨著 ready 設為 1 又會滿足條件回到 idle 狀態，因此代表著 wishbone 的 ack 訊號又隨 ready 設為 0 而變為 0。

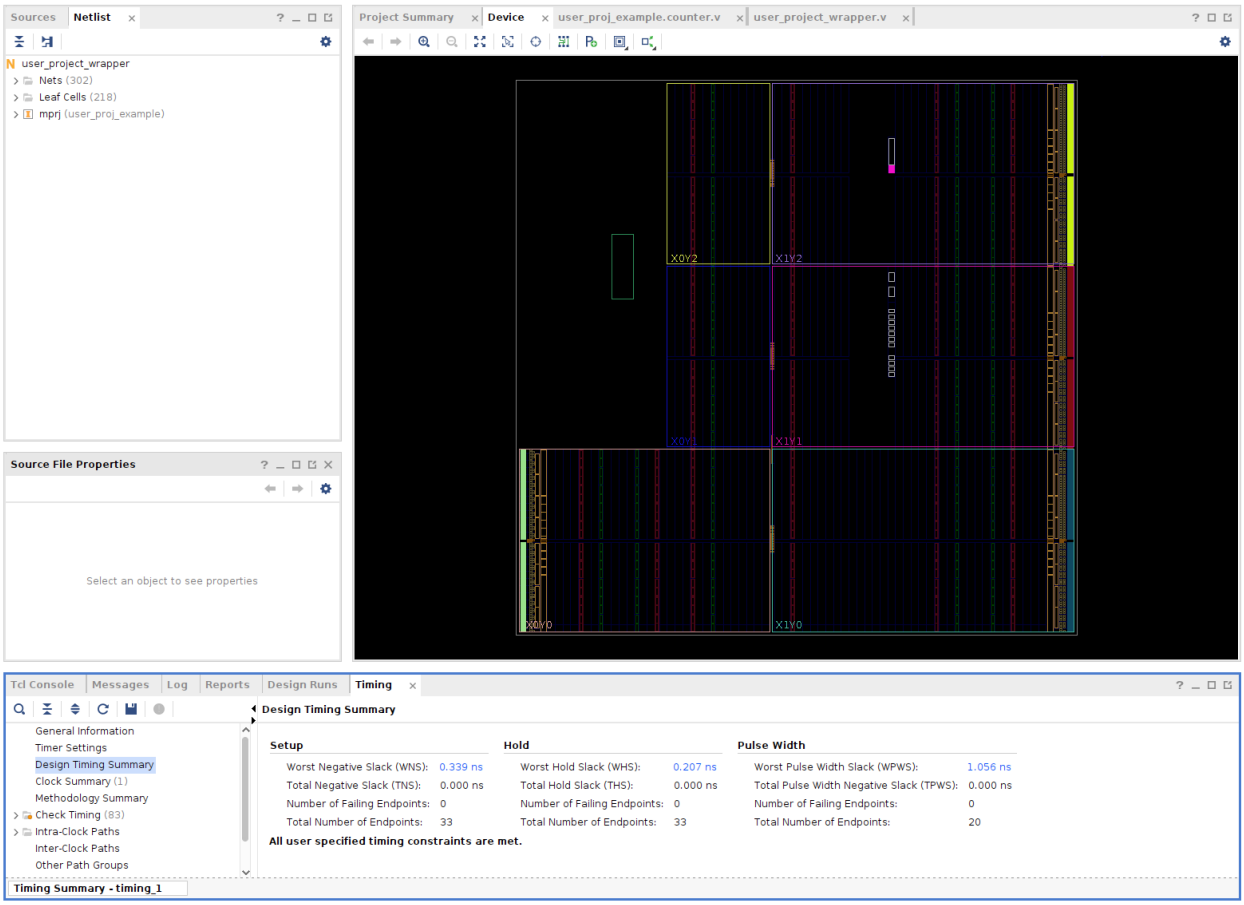
三、 Synthesis report

在 syn.rpt 中能夠看到合成後所使用的各種硬體的數量，其中屬於 LUT 與和 Register 最為重要，在 4-1lab 中所合成的.v 檔只有 user_project_example.counter.v 和 bram.v 兩個檔案，因此 Register 數量只有 17 個以及 30 個 LUT 便能夠運行，而在和成當中有許多要注意的點，其中包含了需要將不包含在 wrapper.v 中的腳位註解掉，再把檔案開頭的 define nettype none 改成 define nettype wire，並且宣告 constraints 時的 axis_clk 應改成 wb_clk_i，在和成完畢後應該檢查 timing report 的 slack 是否為負值，再進而調整 constraints 中的 clk period。

```
1. Slice Logic
--------------

+-------------------------+------+-------+------------+-----------+-------+
|        Site Type        | Used | Fixed | Prohibited | Available | Util% |
+-------------------------+------+-------+------------+-----------+-------+
| Slice LUTs*             |   30 |     0 |          0 |     53200 |  0.06 |
|   LUT as Logic          |   30 |     0 |          0 |     53200 |  0.06 |
|   LUT as Memory         |    0 |     0 |          0 |     17400 |  0.00 |
| Slice Registers         |   17 |     0 |          0 |    106400 |  0.02 |
|   Register as Flip Flop |   17 |     0 |          0 |    106400 |  0.02 |
|   Register as Latch     |    0 |     0 |          0 |    106400 |  0.00 |
| F7 Muxes                |    0 |     0 |          0 |     26600 |  0.00 |
| F8 Muxes                |    0 |     0 |          0 |     13300 |  0.00 |
+-------------------------+------+-------+------------+-----------+-------+
* Warning! The Final LUT count, after physical optimizations and full implement
lower. Run opt_design after synthesis, if not already completed, for a more rea


1.1 Summary of Registers by Type
--------------------------------

+-------+--------------+-------------+--------------+
| Total | Clock Enable | Synchronous | Asynchronous |
+-------+--------------+-------------+--------------+
| 0     |            _ |           - |            - |
| 0     |            _ |           - |          Set |
| 0     |            _ |           - |        Reset |
| 0     |            _ |         Set |            - |
| 0     |            _ |       Reset |            - |
| 0     |          Yes |           - |            - |
| 0     |          Yes |           - |          Set |
| 0     |          Yes |           - |        Reset |
| 0     |          Yes |         Set |            - |
| 17    |          Yes |       Reset |            - |
+-------+--------------+-------------+--------------+
```

```
Max Delay Paths
--------------------------------------------------------------------------------------
Slack (MET) :             0.339ns  (required time - arrival time)
  Source:                 mprj/delayed_count_reg[4]/C
                            (rising edge-triggered cell FDRE clocked by wb_clk_i  {rise@0.000ns fall@
2.000ns period=4.000ns})
  Destination:            mprj/delayed_count_reg[12]/D
                            (rising edge-triggered cell FDRE clocked by wb_clk_i  {rise@0.000ns fall@
2.000ns period=4.000ns})
  Path Group:             wb_clk_i
  Path Type:              Setup (Max at Slow Process Corner)
  Requirement:            4.000ns  (wb_clk_i rise@4.000ns - wb_clk_i rise@0.000ns)
  Data Path Delay:        3.525ns  (logic 1.904ns (54.014%)  route 1.621ns (45.986%))
  Logic Levels:           4  (CARRY4=3 LUT2=1)
  Clock Path Skew:        -0.145ns (DCD - SCD + CPR)
    Destination Clock Delay (DCD):    2.128ns = ( 6.128 - 4.000 )
    Source Clock Delay      (SCD):    2.456ns
    Clock Pessimism Removal (CPR):    0.184ns
  Clock Uncertainty:      0.035ns  ((TSJ^2 + TIJ^2)^1/2 + DJ) / 2 + PE
    Total System Jitter     (TSJ):    0.071ns
    Total Input Jitter      (TIJ):    0.000ns
    Discrete Jitter          (DJ):    0.000ns
    Phase Error              (PE):    0.000ns

    Location             Delay type                Incr(ns)  Path(ns)    Netlist Resource(s)
  -------------------------------------------------------------------    -------------------
```

上圖為 max delay path 的 timing report，在裡面的 data path delay 項能看到此系統所要求的最大延遲為 3.525ns，因此在 constraints 中將 clk period 設為 4ns 能夠滿足系統的需求並且會有較快執行速度。



四、 Other discoveries

在驗證 fir 的功能時，我們將 testbench 中的 mprj 在每次有變化的時候都讀出來，並且手算驗整每一次 fir 計算後輸出的值是否正確，而下圖是在 ubuntu terminal 中的測試資訊，在圖中能看到有三筆 655xx 的數值，代表著計算後的數字為負數，而系統用二補數的方式將資料呈現出來，經過轉換應該為-10、-29、-25，與手動計算結果符合，因此能夠驗證 fir 在這部分能夠正常運作。

```
nter_la_fir$ source run_sim
Reading counter_la_fir.hex
counter_la_fir.hex loaded into memory
Memory 5 bytes = 0x6f 0x00 0x00 0x0b 0x13
VCD info: dumpfile counter_la_fir.vcd opened for output.
checkbits: data = 43840
LA Test 1 started
1 mprj: data =     0
2 mprj: data = 65526
3 mprj: data = 65507
4 mprj: data = 65511
5 mprj: data =    35
6 mprj: data =   158
7 mprj: data =   337
8 mprj: data =   539
9 mprj: data =   732
10 mprj: data =   915
11 mprj: data =  1098
checkbits: data = 43857
LA Test 2 passed
```